Travaux pratiques Services Web REST
TP1 – Introduction

Environnement de travail

Les services seront déployés sous Glassfish, sous la forme d'une archive war (WEB-INF avec fichier web.xml et répertoire classes qui contient les .class des services déployés).
Rappel pour déployer en ligne de commandes :

```
asadmin deploy --force NomArchive.war
```

Configurer votre CLASSPATH pour y ajouter les archives suivantes :

/usr/local/stow/glassfish4/glassfish/modules/javax.ws.rs-api.jar
/usr/local/stow/glassfish4/glassfish/modules/jersey-client.jar
/usr/local/stow/glassfish4/glassfish/modules/jersey-common.jar
/usr/local/stow/glassfish4/glassfish/modules/jersey-guava.jar

Pour tous les exercices, vous placerez vos fichiers dans un package, (sauf pour les exercices spécifiques), et généralement vous ne vous occuperez pas de gérer les cas d'erreur.

Question 1 – Premier service Rest : le traditionnel "Salut" !
Ecrivez un premier service en GET qui renvoie un message de salutation en text/plain.
Renseignez le fichier web.xml, déployez et testez dans un navigateur.
Consultez la description des ressources dans le fichier WADL généré.
Ecrivez ensuite une application cliente Java qui accède au service, dans laquelle vous pouvez afficher la réponse, son statut et le contenu.

Question 2
Ajoutez deux nouvelles méthodes (URL hello et ola) qui renvoient le contenu respectivement en XML et en HTML. Testez dans un navigateur et dans un client lourd Java.

Question 3
Repartez de la question 1 pour ajouter une seconde fonction qui attend un paramètre sur l'URL (path) représentant le nom de la personne, et qui répond un salut personnalisé.

Question 4
Transformez la solution de l'exercice précédent pour utiliser maintenant le passage de paramètres classique en GET sur l'URL (?nom=valeur&autreNom=autreValeur).
Ajoutez une seconde fonction qui gère une valeur par défaut si aucun paramètre n'est transmis.

Travaux pratiques Services Web REST
TP2 – Gestion de ressources

Dans ce TP, on va écrire les bases d'un service qui permet de gérer un carnet de contacts.

Question 1
Ecrivez une classe Contact, nom et numéro 2 Strings, et un constructeur pour initialiser ces 2 valeurs. Proposez les 2 getters et le setter du numéro.
Ecrivez ensuite une classe Carnet qui contient une collection de Contacts. Initialisez cette collection avec 2 contacts.
Ecrivez ensuite un service Annuaire qui gèrera un Carnet. Pour l'instant, ce carnet est une variable de classe, ce afin de conserver les contacts entre 2 utilisations du service (gestion de la persistance plus loin). L'URL `/carnet` permet d'obtenir la liste des contacts. Si le carnet est vide on renvoie "`Liste vide`".
Utilisez dans un navigateur.

Question 2
Ajoutez une méthode qui permet d'obtenir le numéro d'un contact quand on transmet son nom sur l'URL. Si le nom n'existe pas on renvoie le mot "`Inconnu`".
Ecrivez maintenant une application cliente en Java.

Question 3
On veut maintenant pouvoir créer un contact en passant le nom et le numéro en POST.
Si le contact n'existe pas, après la création, renvoyez la chaine "`Contact ... créé`", et dans le cas contraire, renvoyez "`Contact ... déjà existant`".
Utilisez dans l'application cliente.

Question 4
On va maintenant proposer l'utilisation de la fonction de création de contact à partir d'un formulaire HTML.
Dans un premier temps, modifiez web.xml pour associer les ressources Rest à l'URI `/rest`.
Ajouter à l'archive un fichier `nouveau.htm` qui contient un lien d'accès à la ressource de création (donc en get) pour vérifier que l'accès n'est pas possible. Ajoutez le formulaire en post d'accès à la fonction de création.

Question 5
Transformer la fonction de création pour qu'elle retourne l'URI de la ressource Contact qui vient d'être créée. Dans l'application cliente, créez un nouveau contact et réutilisez son URI pour retrouver son numéro.

Question 6
Ajouter une nouvelle ressource en POST qui permet de créer un contact en passant un objet Contact. Pour cela modifiez la classe de façon à ce que le contact devienne un élément XML(annotation @XmlRootElement) et ajoutez un constructeur par défaut.
Pour créer un contact, le client doit connaître la classe Contact ; donc pour cela, vous copierez la classe (fichier .class) dans son répertoire.

Question 7
En utilisant un accès PUT, proposez de modifier un contact en le passant en appel de la fonction. Si le contact n'existe pas ou le carnet est vide, on retourne un contenu vide (pas d'erreur), statut 204.
Utilisez dans l'application cliente.

Question 8
En utilisant un accès DELETE, proposez de supprimer un contact dont on transmet le nom.
Utilisez dans l'application cliente.

Question 9 – Gestion d'une exception – V1
Transformez le service qui trouve un contact de façon à ce qu'il lève une exception si le contact n'existe pas. Pour cela, créez une classe exception ContactNotFoundPlainException.java dont le message sera `"Contact inconnu"`.
Utilisez dans un navigateur : vous constatez que le serveur capture l'exception et génère une page HTML, qui n'est pas exploitable dans une application cliente.

Question 10 – Gestion d'une exception – V2
Ajoutez maintenant une exception ContactNotFoundException.java, qui hérite de WebApplicationException. Définissez une méthode getResponse() qui retourne une réponse avec un status Response.Status.NOT_FOUND et un message `"Ce contact est inconnu"`
Utilisez dans un navigateur puis dans une application client lourd.

Question 11 – Gestion de cookie
Ajoutez un accès GET `/supp/nom` qui supprime une ressource et ajoute un cookie qui correspond au nom de ce contact supprimé. Constatez dans le navigateur.
Proposer un accès `/dernier` qui affiche le dernier supprimé en relisant le cookie.
Ajoutez un accès GET `/listCookies` qui affiche les cookies du domaine (utilisation du contexte et de HttpHeaders).

Question 12 – Persistance
On va maintenant gérer la persistance du carnet de contacts dans un fichier XML : on utilise les annotations et les fonctions de dé/sérialisation (marshal) en XML proposées par l'API JABX.

Question 13 – Introduction à l'authentification
On souhaite maintenant permettre à tout utilisateur authentifié de voir le contenu du carnet, mais seul un administrateur à le droit de supprimer un contact (inutile de travailler sur les autres fonctionnalités). Pour cela, on va gérer des autorisations d'accès suivant des profils d'utilisateur.
Créez des utilisateurs sous Glassfish, ou en utilisant la ligne de commandes :

```
asadmin create-file-user
            --groups appusers:appadmin --authrealmname file Admin
asadmin create-file-user
            --groups appusers --authrealmname file User1
```

où *appusers* et *appadmin* représentent des groupes d'utilisateurs et *Admin* et *User1* des utilisateurs appartenant à ces groupes.
Configurez votre service à travers le fichier `web.xml` et le fichier `glassfish-web.xml`.
Testez dans l'application cliente en utilisant un HttpBasicAuthFilter et en l'enregistrant (register) auprès du Client.